

A bi-criteria path planning algorithm for robotics applications.

Zachary Clawson[†], Xuchu (Dennis) Ding[‡], Brendan Englot[§],
Thomas A. Frewen[‡], William M. Sisson[‡], Alexander Vladimirsky[†]

Abstract: Realistic path planning applications often require optimizing with respect to several criteria simultaneously. Here we introduce an efficient algorithm for bi-criteria path planning on graphs. Our approach is based on augmenting the state space to keep track of the “budget” remaining to satisfy the constraints on secondary cost. The resulting augmented graph is acyclic and the primary cost can be then minimized by a simple upward sweep through budget levels. The efficiency and accuracy of our algorithm is tested on Probabilistic Roadmap graphs to minimize the distance of travel subject to a constraint on the overall threat exposure of the robot. We also present the results from field experiments illustrating the use of this approach on realistic robotic systems.

SECTION 1. INTRODUCTION

Shortest path planning on graphs is among the most studied problems of computer science, with numerous applications ranging from robotics to image registration. Given the node-transition costs, the goal is to find the path minimizing the cumulative cost from a starting position up to a goal state. When costs are non-negative this can be accomplished efficiently by Dijkstra’s algorithm [8] and a variety of related *label-setting methods*.

However, in many realistic applications paths must be evaluated and optimized according to several criteria simultaneously (time, cumulative risk, fuel efficiency, etc). Multiple criteria lead to a natural generalization of path optimality: a path is said to be (weakly) *Pareto-optimal* if it cannot be improved according to all criteria simultaneously. For evaluation purposes, each Pareto-optimal path can be represented by a single point, whose coordinates are cumulative costs with respect to each criterion. A collection of such points is called the *Pareto Front* (PF), which is often used by decision makers in a posteriori evaluation of optimal trade-offs. A related practical problem (requiring only a portion of PF) is to optimize with respect to one (“primary”) criterion only, but with upper bounds enforced based on the remaining (“secondary”) criteria.

It is natural to approach this problem by reducing it to single criterion optimization: a new transition-cost is defined as a weighted average of transition penalties specified by all criteria, and Dijkstra’s method is then used to recover the “shortest” path based on this new averaged criterion. It is easy to show that the resulting path is always Pareto-optimal for every choice of weights used in the averaging. Unfortunately, this *scalarization approach* has a significant drawback [7]: it finds the paths corresponding to convex parts of PF only, while the non-convexity of PF is quite important in many robotics applications.

Numerous generalizations of label-setting methods have also been developed to recover *all* Pareto optimal paths [12, 29, 32, 22, 23, 26, 20, 21]. However, all of these algorithms were designed for general graphs, and their efficiency advantages are less obvious for highly-refined geometrically embedded graphs, where Pareto-optimal paths are particularly plentiful. (This is precisely the case for meshes or random-sampling based graphs intended to approximate the motion planning in continuous domains.) In this paper we describe a simple method for bi-criteria path planning on such graphs, with an efficient approach for approximating the entire PF. The key idea is based on keeping track of the “budget” remaining to satisfy the constraints based on the secondary criterion. This approach was previously used in continuous optimal control by Kumar and Vladimirsky [18]

[†]Center for Applied Mathematics and Department of Mathematics, Cornell University, Ithaca, NY 14853. Both authors’ work was supported in part by the National Science Foundation Grant DMS-1016150. (Emails: zc227@cornell.edu and vlad@math.cornell.edu.)

[‡]United Technologies Research Center, Hartford, CT 06118. These authors’ work was supported by United Technologies Research Center under the Autonomy Initiative. (Emails: xuchu.ding@gmail.com, frewenta@utrc.utc.com, and sissonw2@utrc.utc.com.)

[§]Schaefer School of Engineering & Science, Stevens Institute of Technology, Hoboken, NJ 07030. (Email: benglot@stevens.edu.)

to approximate the discontinuous value function on the augmented state space. More recently, this technique was extended to hybrid systems modeling constraints on reset-renewable resources [6]. In the discrete setting, the same approach was employed as one of the modules in hierarchical multi-objective planners [9]; our current paper extends that earlier conference publication.

The key components of our method are discussed in §2, followed by the implementation notes in §3. In §4 we provide the results of numerical tests on Probabilistic RoadMap graphs (PRM) in two-dimensional domains with complex geometry. Our algorithms were also implemented on a realistic heterogeneous robotic system and tested in field experiments described in §5. Finally, open problems and directions for future work are covered in §6.

SECTION 2. THE AUGMENTED STATE SPACE APPROACH

Consider a directed graph \mathcal{G} on the set of nodes $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{x}_{n+1} = \mathbf{s}\}$ and edges E between nodes. We will choose \mathbf{s} to be a special “source” node (the robot’s current position) and our goal will be to find optimal paths starting from \mathbf{s} . For convenience, we will also use the notation $\mathcal{N}(\mathbf{x}_j) = \mathcal{N}_j$ for the set of all nodes \mathbf{x}_i from which there exists a direct transition to \mathbf{x}_j . We will assume that the cost of each such transition C_{ij} is positive. We begin with a quick review of the standard methods for the classical shortest path problems.

2.1. The single criterion case. Using $\Phi(P)$ to denote the cumulative cost (i.e., the sum of transition costs C_{ij} ’s) along a path P , the usual goal is to minimize Φ over \mathcal{P}_j , the set of all paths from \mathbf{s} to $\mathbf{x}_j \in X$. The standard dynamic programming approach is to introduce the *value function* $U(\mathbf{x}_j) = U_j$, describing the total cost along any such optimal path. Bellman’s optimality principle yields the usual coupled system of equations:

$$U_j = \min_{\mathbf{x}_i \in \mathcal{N}_j} \{C_{ij} + U_i\}, \quad \forall j = 1, 2, \dots, n \quad (1)$$

with $U_{n+1} = U(\mathbf{s}) = 0$. Throughout the paper we will take the minimum over an empty set to be $+\infty$. The vector $U = (U_1, \dots, U_n)$ can be formally viewed as a fixed point of an operator $\mathcal{T} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ defined componentwise by (1). A naive approach to solving this system is to use *value iteration*: starting with an overestimate initial guess $u_0 \in \mathbb{R}^n$, we could repeatedly apply \mathcal{T} , obtaining the correct solution vector U in at most n iterations. This results in $O(n^2)$ computational cost as long as the in-degrees of all nodes are bounded: $|\mathcal{N}_j| < \kappa$ for some constant $\kappa \ll n$. The process can be further accelerated using the Gauss-Seidel relaxation, but then the number of iterations will strongly depend on the ordering of the nodes within each iteration. For acyclic graphs, a standard topological ordering of nodes can be used to essentially decouple the system of equations. In this case, the Gauss-Seidel relaxation converges after a single iteration, yielding the $O(n)$ computational complexity. For a general directed graph, such a *causal* ordering of nodes is a priori unknown, but can be recovered at run time by exploiting the monotonicity of U values along every optimal path. This is the essential idea behind Dijkstra’s classical algorithm [8], which relies on heap-sort data structures and solves this system in $O(n \log n)$ operations. Alternatively, a class of *label-correcting* algorithms (e.g., [3, 5, 4]) attempt to approximate the same causal ordering, while avoiding the use of heap-sort data structures. These algorithms still have the $O(n^2)$ worst-case complexity, but in practice are known to be at least as efficient as Dijkstra’s on many types of graphs [3].

When we are interested in optimal directions from \mathbf{s} to a specific node $\mathbf{t} \in X$ only, Dijkstra’s method can be terminated as soon as $U(\mathbf{t})$ is computed. An A* method [11] can be viewed as a speed-up technique, which further restricts the computational domain (to a neighborhood of (\mathbf{s}, \mathbf{t}) optimal path) using heuristic underestimates of the cost-to-go function. More recent extensions include “Anytime A*” (to ensure early availability of good suboptimal paths) [10, 19, 33] and a number of algorithms for dynamic environments (where some of the C_{ij} values might change before we reach \mathbf{t}) [27, 28, 16].

2.2. Bi-criteria optimal path planning: different value functions and their DP equations.

We will now assume that an alternative criterion for evaluating path quality is defined by specifying “secondary costs” $c_{ij} > 0$ for all the transitions in this graph. Similarly, we define $\phi(P)$ to be the cumulative secondary cost (based on c_{ij} ’s) along P . In this setting, it is useful to consider

several different value functions. The secondary (unconstrained) value function is $V_j = \min_{P \in \mathcal{P}_j} \phi(P)$. It satisfies a similar system of equations: $V_{\mathbf{s}} = 0$ and

$$V_j = \min_{\mathbf{x}_i \in \mathcal{N}_j} \{c_{ij} + V_i\}, \quad j \leq n.$$

A natural question to ask is how much cost must be incurred to traverse a path selected to optimize a *different* criterion. We define \tilde{V}_j as $\phi(P)$ minimized over the set of all primary-optimal paths $\{P \in \mathcal{P}_j \mid \Phi(P) = U_j\}$. Thus, $\tilde{V}_{\mathbf{s}} = 0$, and if we define $\mathcal{N}'_j = \operatorname{argmin}_{\mathbf{x}_i \in \mathcal{N}_j} \{C_{ij} + U_i\} \subset \mathcal{N}_j$, then

$$\tilde{V}_j = \min_{\mathbf{x}_i \in \mathcal{N}'_j} \{c_{ij} + \tilde{V}_i\}, \quad j \leq n.$$

Similarly, we define \tilde{U}_j as $\Phi(P)$ minimized over the set of all secondary-optimal paths $\{P \in \mathcal{P}_j \mid \phi(P) = V_j\}$. Thus, $\tilde{U}_{\mathbf{s}} = 0$, and if we define $\mathcal{N}''_j = \operatorname{argmin}_{\mathbf{x}_i \in \mathcal{N}_j} \{c_{ij} + V_i\} \subset \mathcal{N}_j$, then

$$\tilde{U}_j = \min_{\mathbf{x}_i \in \mathcal{N}''_j} \{C_{ij} + \tilde{U}_i\}, \quad j \leq n.$$

All U_j 's can be efficiently obtained by the standard Dijkstra's method with \tilde{V}_j 's computed in the process as well. The same is true for V_j 's and \tilde{U}_j 's.

Returning to our main goal, we now define the primary-constrained-by-secondary value function $W(\mathbf{x}_j, b) = W_j^b$ as $\Phi(P)$ (the accumulated primary cost) minimized over $\{P \in \mathcal{P}_j \mid \phi(P) \leq b\}$ (the paths with the cumulative secondary cost not exceeding b). We will say that b is *the remaining budget* to satisfy the secondary cost constraint. This definition and the positivity of secondary costs yield several useful properties of W :

- (1) W_j^b is a monotone non-increasing function of b .
- (2) $b < V_j \iff W_j^b = +\infty$.
- (3) $b = V_j \iff W_j^b = \tilde{U}_j$.
- (4) $b \geq \tilde{V}_j \iff W_j^b = U_j$.

We will use the notation

$$\beta(i, b, j) = b - c_{ij} \tag{2}$$

to define the set of feasible transitions on level b :

$$\mathcal{N}_j(b) = \{\mathbf{x}_i \in \mathcal{N}_j \mid \beta(i, b, j) \geq 0\}.$$

The dynamic programming equations for W are then $W_{\mathbf{s}}^b = 0$ for all b and

$$W_j^b = \min_{\mathbf{x}_i \in \mathcal{N}_j(b)} \{C_{ij} + W_i^{\beta(i, b, j)}\}, \quad j \leq n, b > 0. \tag{3}$$

For notational simplicity, it will be sometimes useful to have W defined for non-positive budgets; in such cases we will assume $W_j^b = +\infty$ whenever $b \leq 0$ and $j \leq n$. The Pareto Front for each node \mathbf{x}_j is the set of pairs (b, W_j^b) , where a new reduction in the primary cost is achieved:

$$\mathcal{PF}_j = \{(b, W_j^b) \mid W_j^b < W_j^{b'}, \forall b' < b\}. \tag{4}$$

For each fixed \mathbf{x}_j , the value function W_j^b is piecewise-constant on b -intervals and the entries in \mathcal{PF}_j provide the boundaries/values for these intervals.

Given the above properties of W , it is only necessary to solve the system (3) for $b \in [0, B]$, where B is the *maximal budget level*. E.g., for many applications it might be known a priori that a secondary cumulative cost above some B is unacceptable. On the other hand, if the goal is to recover the entire \mathcal{PF}_j for a specific $\mathbf{x}_j \in X$, we can choose $B = \tilde{V}_j$, or even $B = \max_i \tilde{V}_i$ to accomplish this for all \mathbf{x}_j .

Since all c_{ij} are positive, we observe that the system (3) is *explicitly causal*: the expanded graph on the nodes $\{\mathbf{x}_j^b \mid \mathbf{x}_j \in X, b \geq 0\}$ is acyclic and the causal ordering of the nodes is available a priori. The system (3) can be solved by a single Gauss-Seidel sweep in the direction of increasing b .

2.3. The basic upward-sweep algorithm. For simplicity, we will first assume that all secondary costs are positive and quantized:

$$\exists \delta > 0 \text{ s.t. all } c_{ij} \in \{\delta, 2\delta, 3\delta, \dots\}.$$

This also implies that W_j^b is only defined for $b \in \mathcal{B} = \{0, \delta, 2\delta, \dots, B := m\delta\}$. A simple implementation (Algorithm 1, described below) takes advantage of this structure by storing W_j^b as an array of values for each node $x_j \in X \setminus \{s\}$ and $b \in \mathcal{B}$. For s we will need only one value $W_s = W_s^b = 0$ for all $b \in \mathcal{B}$. For each remaining node $x_j \in X \setminus \{s\}$ with budget $b \in \mathcal{B}$ we can compute the primary-constrained-by-secondary value function W_j^b using formula (3).

	<p>Initialization:</p> <p>1 Compute $U, V, \tilde{U}, \tilde{V}$ for all nodes by Dijkstra's method.</p> <p>2 Set $W_s := 0$</p> <p>3 Main Loop:</p> <p>4 foreach $b = \delta, \dots, B$ do</p> <p>5 foreach $x_j \in X \setminus \{s\}$ do</p> <p>6 if $(U_j < \infty)$ AND $(b \geq V_j)$ then</p> <p>7 if $b = V_j$ then</p> <p>8 $W_j^b := \tilde{U}_j;$</p> <p>9 else</p> <p>10 if $b < \tilde{V}_j$ then</p> <p>11 Compute W_j^b from equation (3);</p> <p>12 else</p> <p>13 $W_j^b := U_j;$</p> <p>14 end</p> <p>15 end</p> <p>16 else</p> <p>17 $W_j^b := +\infty.$</p> <p>18 end</p> <p>19 end</p> <p>20 end</p>
--	--

Algorithm 1: The basic explicitly causal (single-sweep) algorithm.

The explicit causality present in the system is taken advantage of by the algorithm, leading to at most $|X| \cdot |\mathcal{B}| = nm$ function calls to solve equation (3). This results in an appealing $O(\kappa nm)$ complexity, linear in the number of nodes n and the number of discrete budget levels m .

2.4. Quantizing secondary costs and approximating \mathcal{PF} . In the general case, the secondary costs need not be quantized, and even if they are, the resulting number of budget levels m might be prohibitively high for online path-planning. But a slight generalization of Algorithm 1 is still applicable to produce a conservative approximation of W and \mathcal{PF} . This approach relies on “quantization by overestimation” of secondary edge weights with a chosen $\delta > 0$:

$$\hat{c}_{ij} := \delta \left\lceil \frac{c_{ij}}{\delta} \right\rceil \geq c_{ij}. \quad (5)$$

Similarly, we can define $\hat{\phi}(P)$ to be the cumulative quantized secondary cost along P .

The definition of β in (2) is then naturally modified to

$$\beta(i, b, j) = b - \hat{c}_{ij}, \quad (6)$$

with the set of feasible transitions on level $b \in \mathcal{B}$ still defined as $\mathcal{N}_j(b) = \{\mathbf{x}_i \in \mathcal{N}_j \mid \beta(i, b, j) \geq 0\}$. Modulo these changes, the new value function \widehat{W}_j^b is also defined by (3) for all $b \in \mathcal{B}$. It can be similarly computed by Algorithm 1 if the condition on line 7 is replaced by

if $b \in [V_j, V_j + \delta)$ then ...

The above modifications ensure that $\hat{\phi} \geq \phi$ for every path and $W_j^b \leq \widehat{W}_j^b$ for all $b \in \mathcal{B}$, $\mathbf{x}_j \in X$. Moreover, if \widehat{W}_j^b is finite, there always exists some “optimal path” $P \in \mathcal{P}_j$ such that $\hat{\phi}(P) \leq b$ and $\Phi(P) = \widehat{W}_j^b$. Of course, there is no guarantee that such a path P is truly Pareto-optimal with respect to the real (non-quantized) c_{ij} values, but $\hat{\phi}(P) \rightarrow \phi(P)$ and the obtained $\widehat{\mathcal{PF}}_j$ converges to the non-quantized Pareto Front as $\delta \rightarrow 0$. In fact, it is not hard to obtain the bound on $\hat{\phi}(P) - \phi(P)$ by using the upper bound on the number of transitions in Pareto-optimal paths. Let $c = \min_{i,j} c_{ij}$ and $C = \min_{i,j} C_{ij}$. If $k(P)$ is the number of transitions on some Pareto-optimal path P from \mathbf{s} to \mathbf{x}_j , then

$$k(P) \leq K := \min \left(\frac{\tilde{V}_j}{c}, \frac{\tilde{U}_j}{C} \right).$$

Since Algorithm 1 overestimates the secondary cost of each transition by at most δ , we know that

$$\phi(P) \geq \hat{\phi}(P) - K\delta.$$

In the following sections we show that much more accurate “budget slackness” estimates can be also produced at run time.

SECTION 3. IMPLEMENTATION NOTES

A discrete multi-objective optimization problem is defined by the choice of domain Ω , the graph \mathcal{G} encoding allowable (collision-free) transitions in Ω , and the primary and secondary transition cost functions C and c defined on the edges of that graph. Our application area is path planning for a ground robot traveling toward a goal waypoint $\mathbf{x}_j \in X$ in the presence of enemy threat(s). The Pareto Front consists of pairs of threat-exposure/distance values (each pair corresponding to a different path), where any decrease in threat exposure results in an increase in distance traveled and vice-versa.

While recovering the Pareto Front is an important task, deciding how to use it in a realistic path-planning application is equally important. If c is based on the threat exposure and the maximal allowable exposure $B \geq V_j$ is specified in advance, the entire \mathcal{PF}_j is not needed. Instead, the algorithm can be used in a fully automatic regime, selecting the path corresponding to W_j^B . Alternatively, choosing $B = \tilde{V}_j$ we can recover the entire Pareto Front, and a human expert can then analyze it to select the best trade-off between Φ and ϕ . This is the setting used in our field experiments described in Section 5.

3.1. Discretization parameter δ . After the designation of Ω , C , and c , the only remaining parameter needed for Algorithm 1 is $\delta > 0$. For a fixed graph \mathcal{G} , the choice of δ strongly influences the performance:

1. The selection of δ provides direct control over the runtime of the algorithm on \mathcal{G} . Due to the fact that our robot’s sampling-based planning process produces graphs of a predictable size, a fixed number of secondary budget levels m ensures a predictable amount of computation time. Since the complexity is linear in m , the user can use the largest affordable m to define

$$\delta := \tilde{V}_j / m, \quad (7)$$

where \tilde{V}_j is based on the *non-quantized* secondary weights c .

2. The size of δ also controls the coarseness of the approximate Pareto Front. After all, $|\widehat{\mathcal{PF}}_j| \leq m = B/\delta$, and the approximation is guaranteed to be very coarse if the number of Pareto-optimal paths in \mathcal{P}_j is significantly higher.

3. Even if m is large enough to ensure that the number of Pareto optimal paths is captured correctly, there may be still an error in approximating \mathcal{PF}_j due to quantization of secondary costs. For each pair $(b, \widehat{W}_j^b) \in \widehat{\mathcal{PF}}_j$ there is a corresponding path P_b , whose *budget slackness* can be defined as

$$\text{slackness}(P_b) = \hat{\phi}(P_b) - \phi(P_b) = b - \phi(P_b) \geq 0, \quad (8)$$

which can be considered a post-processing technique. Alternatively, if the last transition in this path is from \mathbf{x}_i to \mathbf{x}_j , this can be used to recursively define the slackness measurement

$$S_j^b = \hat{c}_{ij} - c_{ij} + S_i^{b - \hat{c}_{ij}},$$

and compute it simultaneously with \widehat{W}_j^b as an error estimate for $\widehat{\mathcal{PF}}_j$. This approach was introduced in [24].

We point out two possible algorithmic improvements not used in our current implementation:

1. As described, the memory footprint of our algorithm is proportional to nB/δ since all W values are stored on every budget level. In principle, we need to store only finite values larger than U_j ; for each node \mathbf{x}_j , their number is $m(j) = (\tilde{V}_j - V_j)/\delta$. This would entail obvious modifications of the main loop of Algorithm 1 since we would only need to update the “still constrained” nodes:

$$\text{Still_Constrained}(b) = \left\{ \mathbf{x}_j \in X \setminus \{\mathbf{s}\} \mid b \in [V_j, \tilde{V}_j) \right\}.$$

This set can be stored as a linked list and efficiently updated as b increases, especially if all nodes are pre-sorted based on \tilde{V}_j ’s.

2. If the computational time available only permits for a coarse budget quantization (i.e. δ is large), an initial application of Algorithm 1 may result in failure to recover a suitable number of paths from a set of evenly-spaced budget levels. Large differences in secondary cost often exist between paths that lie in different homotopy classes, and individual homotopy classes may contain a multitude of paths with small differences in secondary cost. To remedy this, δ might be refined adaptively, for a specific range of budget values. The slackness measurements can be employed to determine when such a refinement is necessary.

3.2. Non-monotone convergence: a case study. In real-world path planning examples, the true secondary transition costs c_{ij} are typically not quantized, and equation (5) introduces a small amount of error dependent on δ . These errors will vanish as $\delta \rightarrow 0$, but somewhat counterintuitively the convergence is generally not monotone.

This issue can be illustrated even in single criterion path-planning problems. Consider a simple graph in Fig. 1a, where there are two paths to travel from \mathbf{x}_0 to \mathbf{x}_2 . If the edge-weights for this graph were quantized with formula (5), an upper bound for a given path would be:

$$\hat{\phi}(P) \leq \phi(P) + \delta \cdot k(P).$$

Figs. 1b & 1c each show the quantized cost along each path along with this upper bound. There are several interesting features that this example illustrates. First, the convergence of the quantized edge weights is non-monotone, as expected. Further, as $\delta \rightarrow 0$, the minimum-cost path corresponding to the quantized edge-weights switches between the two feasible paths in the graph. Fig. 2 shows the graph with quantized weights \hat{c} for specific values of δ , where the path shown in bold is the optimal quantized path. Finally, as δ decreases, a threshold ($\delta = 0.1$) is passed where the quantized cost along the top (truly optimal) path is always less than the quantized cost along the bottom (suboptimal) path.

Returning to the bi-criteria planning, the monotone decrease of errors due to quantization can be ensured if every path P satisfying $\hat{\phi}(P) \leq b$ for a fixed value of δ will still satisfy the same inequality as δ decreases. If we define a sequence $\delta_k = B/m_k$, the simplest sufficient condition is to use $m_{k+1} = 2m_k$. This is the approach employed in all experiments of Section 4. For the rest of the paper we suppress the hats on the W ’s for the sake of notational convenience; the slackness measurements corresponding to each value of δ are shown for diagnostic purposes in all experiments of Section 4.

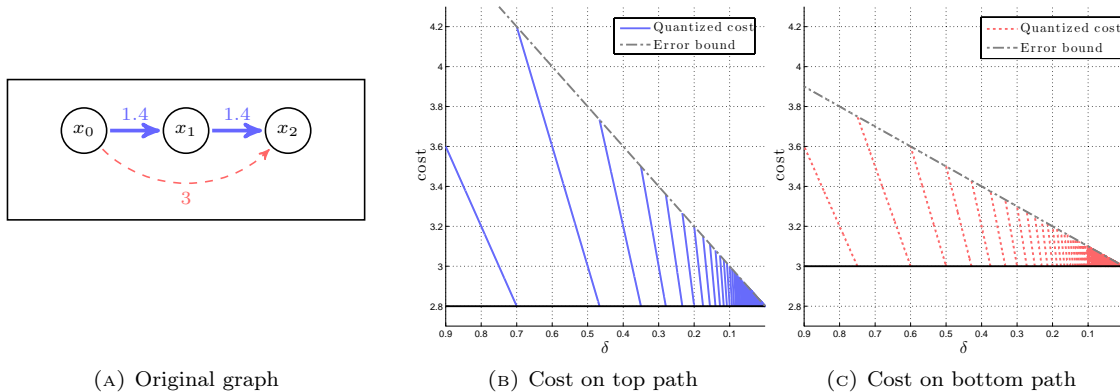


FIGURE 1. (A) Graph with two feasible paths from x_0 to x_2 . (B) & (C) Quantized cost along the top and bottom paths (respectively) as δ decreases (left-to-right).

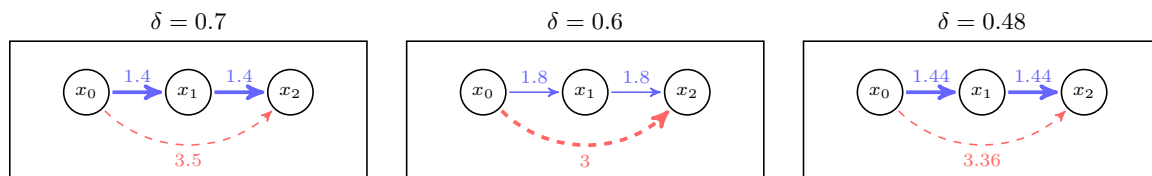


FIGURE 2. The graph in Figure 1a with quantized edge-weights for specific values of δ (decreasing from left-to-right). The optimal path in each subfigure is in bold.

SECTION 4. BENCHMARKING ON SYNTHETIC DATA

We aim to gain further understanding of the proposed algorithm in applications of robotic path planning by focusing most of the attention on a simple scenario. Here, in contrast to Section 5, our goal is to study the convergence properties of the algorithm by refining two accuracy parameters: the number of PRM nodes in the graph and the number of budget levels (determined by δ). Computations are done offline on a laptop rather than a robotic system, allowing for more computational time than in a real scenario. We assume that the occupancy map and threats are known a priori so that we expend no effort in learning this information. All tests are conducted on a laptop with an Intel i7-4712HQ CPU (2.3GHz quad-core) with 16GB of memory.

4.1. Case Study Setup. The results presented in this paper require a graph that accurately represents the motion of a robot in an environment with obstacles, where nodes correspond to collision-free configurations and edges correspond to feasible transitions between these configurations. We are particularly interested in graphs that contain a large number of transitions between any two nodes. There are a number of algorithms capable of generating such graphs, such as the Rapidly-exploring Random Graphs (RRG) [14] or Probabilistic Roadmap (PRM) [15] algorithms. In this work we chose to use the PRM algorithm as implemented in Open Motion Planning Library (OMPL) [31] as a baseline algorithm to generate the roadmap/graph. In essence, the PRM algorithm generates a connected graph representing the robot's state (e.g. position, orientation, etc...) in the environment. The graph is generated by randomly sampling points in the state-space of the robot (called the configuration space in robotics) and trying to connect existing vertices of the graph within the sampled configuration.

In case studies examined, we construct a roadmap as a directed graph $\mathcal{G} = (X, E)$ where X is the set of nodes and E is the set of transitions (edges). The nodes of \mathcal{G} represent the positions of the robot in 2D, i.e. $\mathbf{x} \in X \subset \mathbb{R}^2$, and an edge $e = (\mathbf{x}_i, \mathbf{x}_j) \in E$, $\mathbf{x}_i, \mathbf{x}_j \in X$ represents that there is a collision-free line segment from node \mathbf{x}_i to node \mathbf{x}_j in the PRM graph. Each edge of \mathcal{G} is assigned a primary and secondary transition cost by two cost functions $C, c : E \rightarrow \mathbb{R}^+$, respectively.

For the configuration space we chose $\Omega \subset \mathbb{R}^2$ to be sampled by the OMPL planning library [31] with the default uniform (unbiased) sampling, and two slight modifications: we increased the number of points along each edge that are collision-checked for obstacles and removed self-transitions between nodes. For real-world examples the roadmap \mathcal{G} is typically generated by running the PRM algorithm for a fixed amount of time. However, for testing purposes we specify the number of nodes desired in \mathcal{G} .

Two test cases are presented:

- Sections 4.3–4.5 present an example that uses an occupancy map generated by the Hector SLAM algorithm [17] via a LIDAR sensor in a real testing facility.
- Section 4.6 presents a more complicated scenario in a synthetic environment to illustrate the much broader scope of the algorithm.

Both tests assume that the physical dimensions of Ω are $450\text{m} \times 450\text{m}$ (so each pixel of the occupancy map is 1m^2). Each edge is generated by connecting vertices in the graph that are ‘close together’ via straight-line segments and collision-checking the transition (edge) with a sphere of radius 5m, which represents the physical size of the robot.

The first occupancy map considered is shown in Fig. 3a; the color black represents occupied areas (walls) whereas white represents the unoccupied areas (configuration space). The concentric circles represent the contours of a threat level function (to be defined). A PRM-generated roadmap with 2,048 nodes and 97,164 edges over this occupancy map is shown in Fig. 3b.

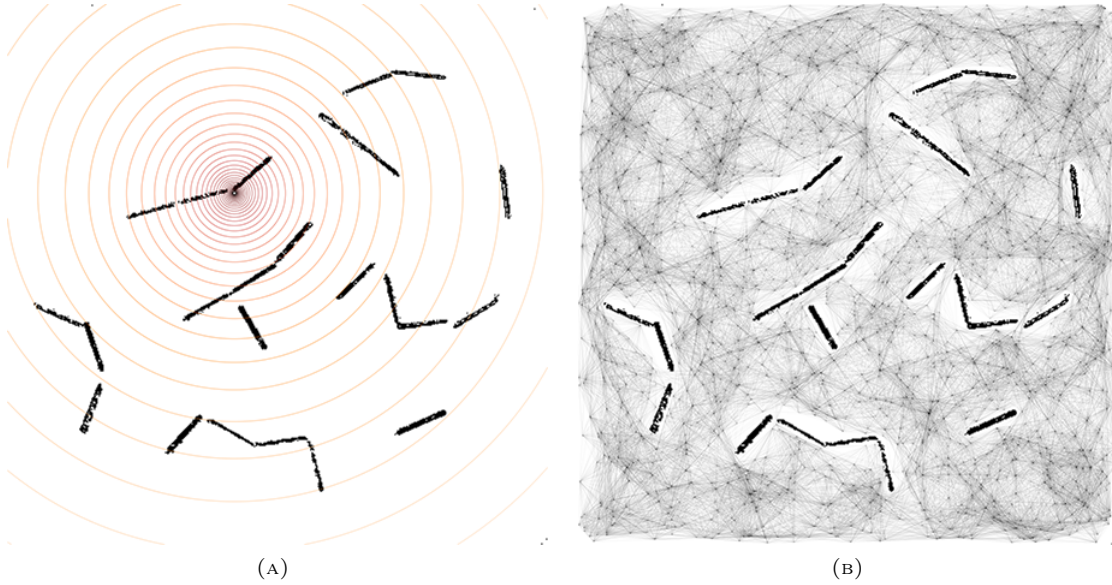


FIGURE 3. (A) Environment with obstacles and contours of a threat exposure function.
(B) PRM Roadmap with 2,048 nodes and 97,164 edges.

4.2. Cost Functions. We consider two cost functions D and T for the multi-objective planning problem. The cost function D represents the Euclidean distance between two adjacent nodes in X , i.e. $D(e) = \|\mathbf{x}_i - \mathbf{x}_j\|$ where $e = (\mathbf{x}_i, \mathbf{x}_j)$.

The cost function T denotes the threat exposure along an edge in the graph. To define T we assume that there are threats $\mathcal{K} = \{1, 2, \dots, N_T\}$ in the environment, which are indicated by their position $\mathbf{p}_k \in \mathbb{R}^2$, severity s_k , minimum radius $r_k \geq 0$, and visibility radius $R_k \geq 0$ for each $k \in \mathcal{K}$. For each threat, these parameters define a threat exposure function

$$\tau_k(\mathbf{x}) = \begin{cases} s_k/r_k^2 & \text{if } \|\mathbf{x} - \mathbf{p}_k\| \leq r_k \\ s_k/\|\mathbf{x} - \mathbf{p}_k\|^2 & \text{if } \|\mathbf{x} - \mathbf{p}_k\| \in (r_k, R_k) , \\ s_k/R_k^2 & \text{if } \|\mathbf{x} - \mathbf{p}_k\| \geq R_k \end{cases} \quad (9)$$

for $\mathbf{x} \in \mathbb{R}^2$. Fig. 3a shows the contours of τ for a single threat with $s = 20$, $r = 5\text{m}$, and $R = +\infty$. For simplicity we will always assume that $R_k = +\infty$ for all $k \in \mathcal{K}$ in all examples.

The cumulative threat exposure function is then defined as the integral of the instantaneous threat exposure along the transition (edge), summed over all threats $k \in \mathcal{K}$

$$T(e) = \sum_{k \in \mathcal{K}} \int_0^1 \tau_k [(1-t) \cdot \mathbf{x}_i + t \cdot \mathbf{x}_j] \|\mathbf{x}_j - \mathbf{x}_i\| dt \quad (10)$$

where $e = (\mathbf{x}_i, \mathbf{x}_j) \in E$ is parameterized above by assuming the robot moves at a constant speed. Note that in (10), we penalize both the proximity to each threat and the duration of exposure. An example threat placement in the environment is shown in Fig. 3a along with the associated contours of the threat exposure function.

4.3. Pareto Front. We use the proposed multi-objective planning algorithm to identify paths that lie on the Pareto front of the primary and secondary cost. Our approach enables the re-use of a roadmap for searches under different objectives and constraints.

We rely on the two cost functions D and T described in Section 4.2. In every Pareto Front shown we will reserve the horizontal axis for accumulated values of T and the vertical axis for accumulated values of D . We first point out a Pareto Front in Fig. 4a that is generated for the environmental setup shown in Fig. 3. The Pareto Front is color-coded (dark blue to magenta) indicating the cumulative threat exposure T along the path (low-to-high), where each color-coded Pareto-optimal path within the configuration space is shown in Fig. 4b. As explained in Section 3 there is additional error due to the quantization of the secondary costs. As discussed, one method for measuring this error is to calculate the *true secondary cost* along each path, which is shown by the gray markers in every Pareto Front plot.

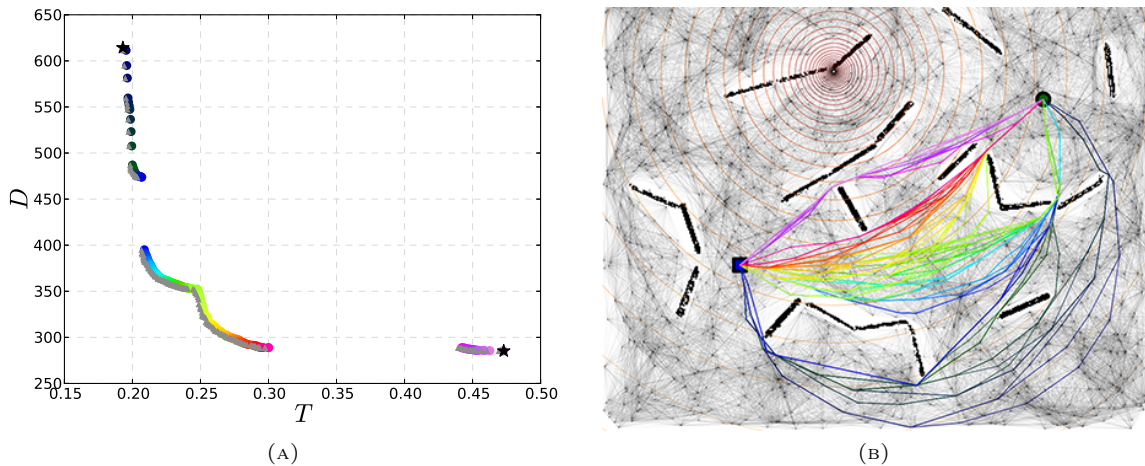


FIGURE 4. Test case with $C = D$ and $c = T$ and the same parameters as Figure 3. (A) Pareto Front with a strong non-convexity. (B) Pareto-optimal paths in the configuration space, color-coded to correspond with the Pareto Front.

4.4. Effect of discretization δ . As discussed in Section 2, the parameter δ determines the ‘discretization’ of the secondary budget allowance. For all results we use $\delta = \tilde{V}_j/m$ as defined in Equation (7) in Section 3.

In Fig. 5 we generate a high-density 40,000-node graph (with 3,030,612 edges) and observe the effect of decreasing δ (increasing m) on the results. As δ decreases, it is clear that the produced Pareto Front approaches the true secondary cost curve in gray. The geometric sequence of m values used to generate the Pareto Fronts in Figs. 5a–5d is important as discussed in Section 3.2 – it is what guarantees the monotone convergence. We note that even for non-geometric sequences of m that non-monotone convergence is difficult to visually observe.

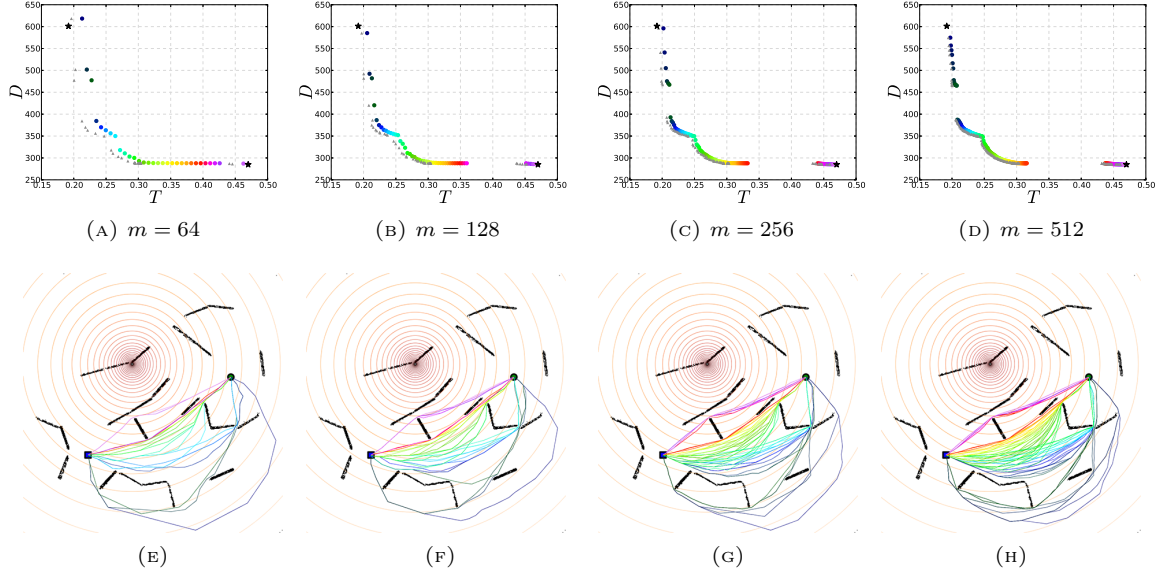


FIGURE 5. Test case with $C = D$ and $c = T$. Results correspond to a 40,000 node graph with 3,030,612 edges. (A) – (F): PF and paths as m (number of budget levels) varies. Results are qualitatively the same for $m > 512$.

4.5. Swapping primary/secondary costs. Throughout this paper we have assumed that the roles of primary and secondary edge weights are fixed, focused on realistic planning scenarios minimizing the distance traveled subject to constraints on exposure to enemy threat(s). However, an equivalent solution may also be obtained by switching the costs, with a better approximation of \mathcal{PF} sometimes obtained without increasing the number of budget levels [24].

Our experimental evidence indicates that the algorithm does a better job of recovering the portion of the Pareto Front where the slope of the front has smaller magnitude. In other words, small changes in secondary budget result in small changes in accumulated primary cost, e.g. see Fig. 5. Fig. 6 shows the result of setting the primary cost to threat exposure ($C = T$) and the secondary cost to be distance ($c = D$). This algorithmic feature can be used to recover a more uniformly dense Pareto Front without significantly increasing the computational cost: consider collating the right portion of Fig. 5b and the left part of Fig. 6b.

4.6. Visibility from enemy observer. Previous results focused on an example with a single enemy observer with a simple model for threat exposure. In this final subsection we illustrate the algorithm on a much broader example including multiple enemies with limited visibility behind obstacles. We assume that the size of this domain Ω is 450m \times 450m, and suppose there are two enemy observers with parameters:

Enemy 1	$s_1 = 20$	$\mathbf{p}_1 = (225\text{m}, 292.5\text{m})$	$r_1 = 5\text{m}$	$R_1 = +\infty$
Enemy 2	$s_2 = 5$	$\mathbf{p}_2 = (225\text{m}, 180\text{m})$	$r_2 = 5\text{m}$	$R_2 = +\infty$

For this problem we take the visibility of enemies into account in the construction of a new threat function T^{vis} . We calculate T^{vis} just as in (10), but modify the individual threat equations $\tau_k(\mathbf{x})$ to account for limited visibility. We define modified threat level functions

$$\tau_k^{vis}(\mathbf{x}) = \begin{cases} \tau_k(\mathbf{x}) & \text{if the line segment from } \mathbf{x} \text{ to } \mathbf{p}_k \text{ is collision-free} \\ \epsilon & \text{otherwise,} \end{cases}$$

for some small $\epsilon > 0$ (we found $\epsilon = 1/\text{Area}(\Omega)$ to work well).

In Fig. 7a we show the environment where, just as before, the black rectangles represent obstacles that define the configuration space. The contours of the summed threat exposure functions, $\tau_1^{vis}(\mathbf{x}) + \tau_2^{vis}(\mathbf{x})$, are plotted. The locations of the two enemies are visually apparent, and the visibility of

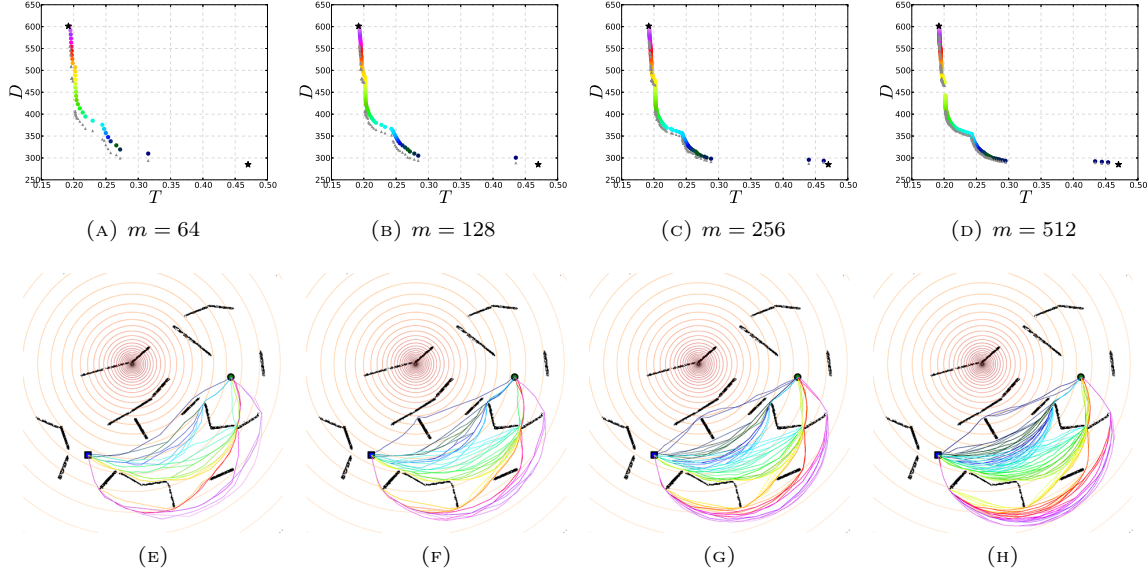


FIGURE 6. Test case with $C = T$ and $c = D$. The same experimental setup as Fig. 5 (except with C and c swapped). The Pareto Front is plotted above the “true secondary cost curve” in this figure only (rather than to the right), due to the secondary objective being distance (vertical axis).

each enemy can also be easily seen. In particular, the white regions in the image show the points $\mathbf{x} \in \Omega$ where visibility is obstructed for both enemies, i.e. $\tau_k^{vis}(\mathbf{x}) \equiv \epsilon$ for all $k = 1, 2$. Fig. 7b shows the roadmap generated for a 2,048 node graph with 103,672 edges in this environment.

Figs. 7c & 7d show the results with the settings $C = D$ and $c = T^{vis}$. Fig. 7c shows the results of the algorithm using the same roadmap (Fig. 7b) with $m = 2,048$. The corresponding Pareto Front in Fig. 7d exhibits strong non-convexity on this particular domain and generated graph due to the large number of obstacles.

Our final test is to use this visibility example to study the convergence of the Pareto Front as the number of nodes in the graph increases. We fix $m = 2048$ budget-levels in order to accurately capture the Pareto Front PF_n corresponding to the generated graph of n nodes. We plot PF_n in Fig. 8 as n varies. There is some Pareto Front PF corresponding to a ‘continuous’ version of this multi-objective problem, and we see that as n increases PF_n converges to PF .

SECTION 5. EXPERIMENTAL DATA

The multi-objective path planning algorithm described in this paper was implemented and demonstrated on a Husky ground vehicle produced by Clearpath Robotics [1]. A picture of the vehicle used in the experiments is shown in Figure 9a. The vehicle was equipped with a GPS receiver unit, a WiFi interface, a SICK LMS111 Outdoor 2D LIDAR [2], an IMU, wheel encoders and a mini-ITX single board computing system with a 2.4GHz Intel i5-520M processor and 8GB of RAM.

Although the GPS unit is used for localization, the associated positional uncertainty typically cannot meet requirements needed for path planning. Moreover, a map of the environment is required to generate a roadmap to support path planning as discussed in the previous section. To this end, we use a Simultaneous Localization and Mapping (SLAM) algorithm for both localization and mapping, and in particular, the Hector SLAM open source implementation [17]. Hector SLAM is based on scan matching of range data which is suitable for a 2D LIDAR such as the SICK LMS111. The output of Hector SLAM is an occupancy grid representing the environment and a pose estimate of the vehicle in the map. Details of the algorithm and implementation can be found in reference [17]. In order to allow stable and reliable state estimation, we also implemented an extended Kalman filter (EKF). Position estimates were computed by fusing inertial information and SLAM pose estimates through the EKF.

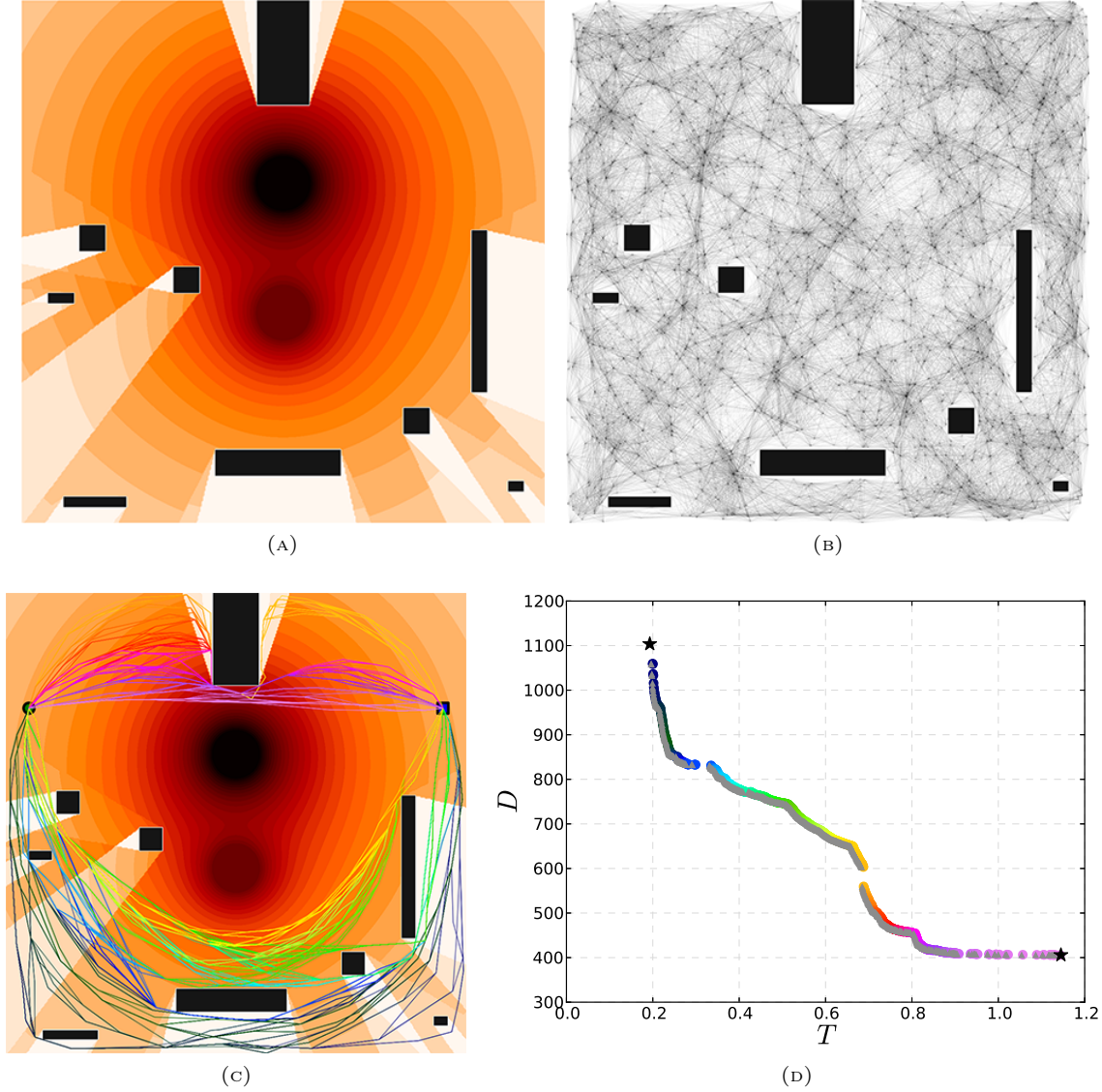


FIGURE 7. Test case with $C = D$ and $c = T^{vis}$. (A) Obstacles plotted with a contour map of the threat exposure (with visibility). (B) Roadmap. (C) & (D) The paths and Pareto Front corresponding to this problem setup, where $m = 2,048$.

A set of 24 “Jersey barriers” placed on flat ground were used to create a complex maze-like environment with multiple interconnecting corridors. Figure 9b shows the barriers and their positions on the ground. The map generated by Hector SLAM for the obstacle course is shown in Figure 9c. The map of the environment is known to the vehicle before the start of the multi-objective planning mission. We utilized the Open Motion Planning Library (OMPL) [30] which contains implementations of numerous sampling-based motion planning algorithms. We have prior experience using this library for sampling-based planner navigation of an obstacle-rich environment [13]. The generality of OMPL facilitated the development and implementation of our field-capable bi-criteria path planning algorithm: efficient low-level data structures and subroutines are leveraged by OMPL (such as the Boost Graph Library (BGL) [25]) and several predefined robot types (and associated configuration spaces) are available. For the purposes of Husky path planning we apply the PRM algorithm within an \mathbb{R}^2 state space.

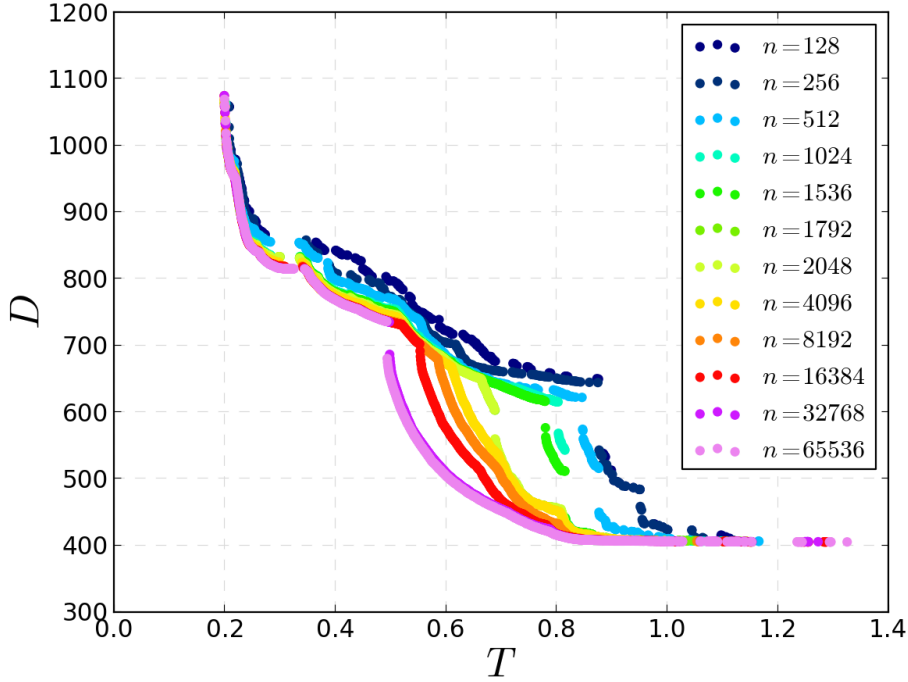


FIGURE 8. Test case with $C = D$ and $c = T^{vis}$. Using the same setup as Fig. 7, we study the effect of increasing the number of nodes n in the graph. The budget levels were finely discretized using $m = 2048$.

For the experiments provided in this section, we set the number of budget levels to $m = 768$. The roadmap “grow-time” is set to be about 0.25 seconds, and planning boundary to be a box of size $25m \times 25m$, resulting a roadmap with about $n = 8,000$ vertices and 116,000 edges (e.g., see Fig. 10b). The time to generate the roadmap graph and assign edge-weights is about 1 second, and time to compute the solution and generate the Pareto Front is about 1 second. Thus, a fresh planning instance from graph generation to computing a solution takes about 2 seconds. We also implemented a number of features in the planner intended to add robustness and address contingencies that may arise during the mission, such as when threat values in the environment change. In this case, the execution framework will attempt to re-plan with updated threat values. Since the graph need not be re-generated, replanning is typically much faster (about 1 second).

In the subsequent mission, we assume the Husky is tasked with navigating from its current location to a designated goal waypoint, while avoiding threat exposure to a set of fixed and given threats, i.e., we aim to minimize distance (primary cost) subject to a maximum allowable level of threat exposure (secondary cost). We use the cost functions D (distance) and T (threat exposure) as given in Section 4.2. For threat exposure, we use a single threat with $s = 1$, $r = 0$ and $R = +\infty$. Once the path is computed, the path is sent down to the lower level path smoothing module. The path smoother is part of the hierarchical planning and execution framework as discussed in [9] and is implemented using model predictive control with realistic vehicle dynamics.

Since threat exposures are difficult to quantify absolutely but easy to compare relatively, we designed a Graphical User Interface (GUI) to pick a suitable path from the Pareto front reconstructed by the proposed algorithm, i.e., the user chooses the shortest path subject to an acceptable amount risk from among the paths in the Pareto front. The process to execute the planning mission is illustrated in Fig. 10 can be described as follows:

1. Given an occupancy map of the environment generated by the Hector SLAM algorithm and the known threat location, the user sets the desired goal location for the mission (see Fig. 10a). The user has some predefined level of tolerance/budget for the amount of threat exposure allowed.

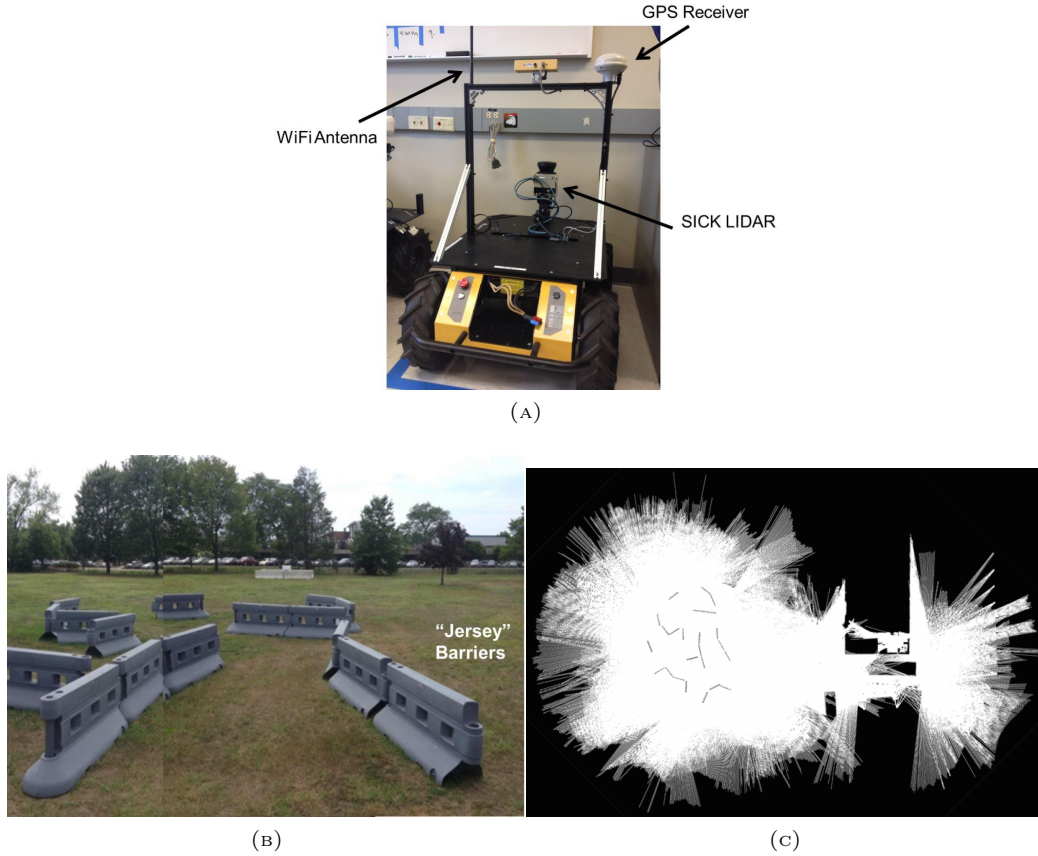


FIGURE 9. (A) Husky robot from ClearPath Robotics. The GPS unit, WiFi antenna and SICK planar LIDAR are shown. A camera is also visible on the roll bar, but is was not used in the experiments. (B) Complex obstacle course created with “Jersey barriers”. (C) Map generated by Hector SLAM for the obstacle course.

- Say, for example, a threat tolerance of 0.79 is permitted in the example shown in Figure 10 (for reference, the shortest distance path has a threat exposure of 11).
2. An initial PRM is generated with edge-weights computed by primary and secondary cost functions. Algorithm 1 is executed and the results are used to compute the Pareto Front (see Fig. 10b). Next, the path corresponding to each point on the PF is recovered and shown in the GUI, where color indicates the amount of secondary cost (see Fig. 10c).
 3. From the paths available in Fig. 10c and the amount of threat exposure tolerated, an expert analyzes the results. Originally the budget was defined to be 0.79, which corresponds to a path with a length of 32.5m (primary cost). The expert realizes if the exposure allowance is slightly increased to 0.81, a path with length 25.25m is available. These two points correspond to the large vertical drop in the Pareto Front in Fig. 10.
 4. From the results and analysis, the user then chooses a path by clicking on a dot in the lower right plot. The picked path is highlighted in bold (see Fig. 10c). The user then clicks a button in the GUI to verify the path as desired (see Fig. 10d), and finally the selected path is sent to lower level modules for path smoothing and vehicle control.

Fig. 11 shows four representative snapshots from a real mission in progress.

SECTION 6. CONCLUSIONS

We have introduced an efficient and simple algorithm for bi-criteria path planning. Our method has been extensively tested both on synthetic data and in the field, as a component of a real robotic

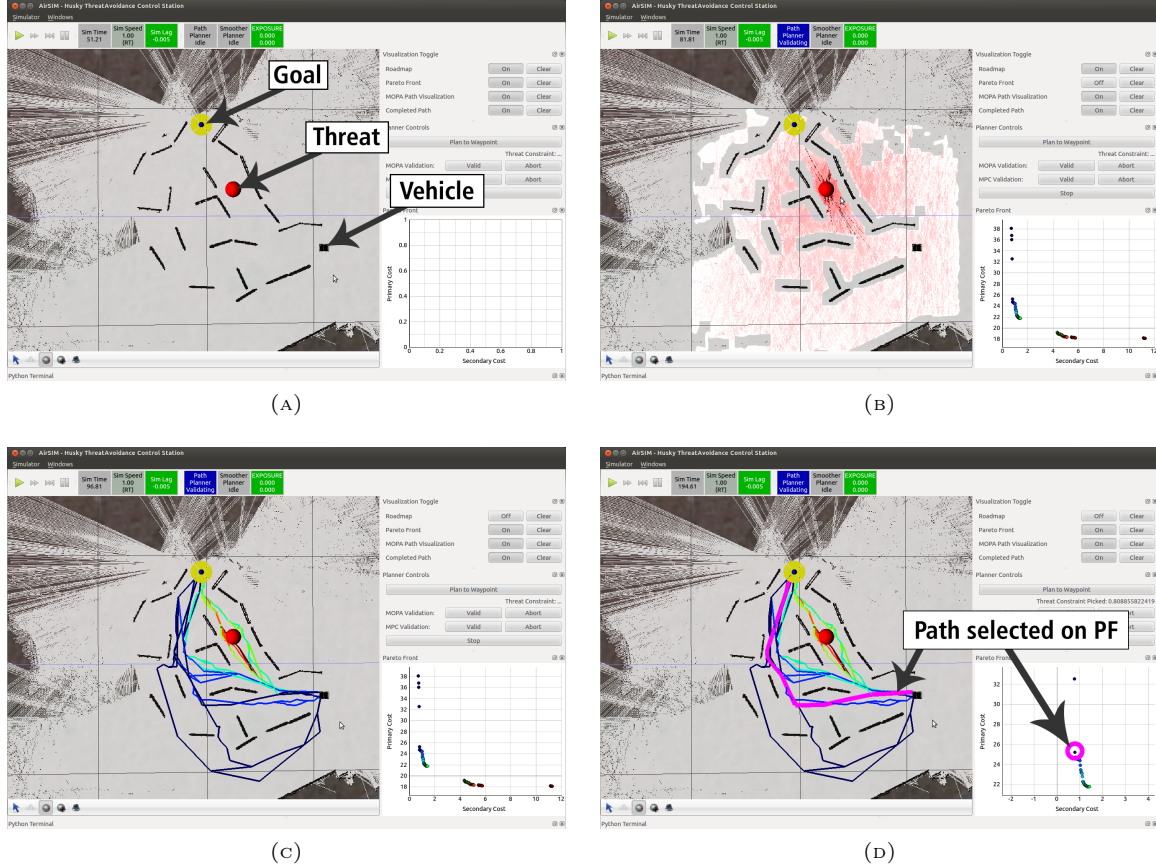


FIGURE 10. (A) The user sets the goal by moving the marker to a location in the occupancy map. The vehicle and threat locations are shown. (B) A roadmap is generated with edges colored according to the secondary cost (threat exposure). The algorithm is run and the Pareto Front is visualized where the vertical and horizontal axes are primary and secondary costs, respectively. (C) The Pareto-optimal paths are shown, each corresponding to a dot on the Pareto Front (by color association). (D) The expert user clicks on a dot in the plot in the lower right corner which highlights the corresponding path in the GUI, choosing a path based on the desired trade-off between primary and secondary cost. The Pareto Front has been zoomed-in for this subfigure.

system. Unlike prior methods based on scalarization, our approach recovers the entire Pareto Front regardless of its convexity. As an additional bonus, choosing a larger value of δ boosts the efficiency of the method, resulting in an approximation of PF, whose accuracy can be further assessed in real time.

The computational cost of our methods is $O(nm)$ corresponding to the number of nodes in the budget-augmented graph. Of course, the total number p of nodes with distinct Pareto optimal cost tuples can be much smaller than nm . In such cases, the prior label-setting algorithms for multi-objective planning will likely be advantageous since their asymptotic cost is typically $O(p \log p)$. However, in this paper we are primarily interested in graphs used to approximate the path planning in continuous domains with edge costs correlated with the geometric distances. As we showed in Section 4, for such graphs the number of points on PF is typically quite large (particularly as PRM graphs are further refined), with new Pareto optimal paths becoming feasible on most of the budget levels. Coupled with domain restriction techniques and the simplicity of implementation, this makes our approach much more attractive.

Several extensions would obviously greatly expand the applicability of our method. We hope to extend it to a higher number of simultaneous criteria and introduce A^* , D^* , and “anytime planning”

versions. In addition, it would be very useful to develop a priori bounds for the errors introduced in the PF as a function of δ . Another direction is automating the choice of primary/secondary cost to improve the method's efficiency.

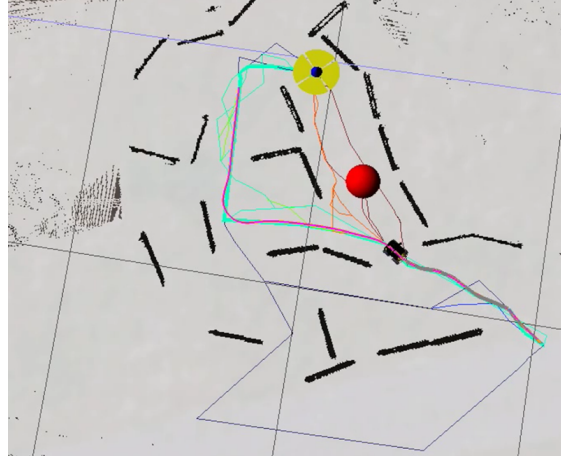
Acknowledgements. The authors would like to acknowledge and thank the whole Autonomy team at United Technologies Research Center. In particular, the authors would like to thank Suresh Kannan, Alberto Speranzon, and Andrzej Banaszuk for productive discussions of the algorithm and implementations of the simulation software. In addition, the authors would like to thank Amit Surana, Zohaib Mian, and Jerry Ding for the support from the Autonomy Initiative and help during field experiments. The authors are also grateful to Blane Rhoads for his suggestions on testing the slackness of the Pareto Front approximation.

REFERENCES

1. Clearpath robotics - husky ground vehicle.
2. Sick 2d lidar.
3. Dimitri P Bertsekas, *A simple and fast label correcting algorithm for shortest paths*, Networks **23** (1993), no. 8, 703–709.



(A)



(B)



(C)



(D)

FIGURE 11. At left, images of the experiment; at right, details of the monitoring station showing the status of the vehicle (the black rectangle), the threat location (the red dot), and the optimal trajectories computed by the algorithm. (A) and (B) are the vehicle at near start of the mission and (C) and (D) are the vehicle near the end of the mission.

4. Dimitri P. Bertsekas, *Dynamic programming and optimal control*, vol. 1, Athena Scientific Belmont, MA, 1995.
5. Dimitri P Bertsekas, Francesca Guerriero, and Roberto Musmanno, *Parallel asynchronous label-correcting methods for shortest paths*, Journal of Optimization Theory and Applications **88** (1996), no. 2, 297–320.
6. W. Chen, Z. Clawson, S. Kirov, R. Takei, and A. Vladimirovsky, *Optimal control with budget constraints and resets*, SIAM Journal on Control and Optimization **53** (2015), 712–744.
7. I. Das and J.E. Dennis, *A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems*, Struct. Optim. **14** (1997), 63–69.
8. Edsger W Dijkstra, *A note on two problems in connexion with graphs*, Numerische mathematik **1** (1959), no. 1, 269–271.
9. Xuchu Ding, Brendan Englot, Allan Pinto, Alberto Speranzon, and Amit Surana, *Hierarchical multi-objective planning: From mission specifications to contingency management*, Robotics and Automation (ICRA), 2014 IEEE International Conference on, IEEE, 2014, pp. 3735–3742.
10. Eric A Hansen, Shlomo Zilberstein, and Victor A Danilchenko, *Anytime heuristic search: First results*, Univ. Massachusetts, Amherst, MA, Tech. Rep **50** (1997).
11. Peter E Hart, Nils J Nilsson, and Bertram Raphael, *A formal basis for the heuristic determination of minimum cost paths*, Systems Science and Cybernetics, IEEE Transactions on **4** (1968), no. 2, 100–107.
12. J. M. Jaffe, *Algorithms for finding paths with multiple constraints*, Networks **14** (1984), 95–116.
13. S. K. Kannan, W. M. Sisson, D. A. Ginsberg, J. C. Derenick, X. C. Ding, T. A. Frewen, and H. Sane, *Close proximity obstacle avoidance using sampling-based planners*, 2013 AHS Specialists’ Meeting on Unmanned Rotorcraft and Network-Centric Operations, 2013.
14. Sertac Karaman and Emilio Frazzoli, *Sampling-based algorithms for optimal motion planning*, The International Journal of Robotics Research **30** (2011), no. 7, 846–894.
15. Lydia E. Kavraki, Petr Svestka, J-C Latombe, and Mark H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, IEEE Transactions on Robotics and Automation **12** (1996), no. 4, 566–580.
16. Sven Koenig and Maxim Likhachev, *Fast replanning for navigation in unknown terrain*, Robotics, IEEE Transactions on **21** (2005), no. 3, 354–363.
17. Stefan Kohlbrecher, Johannes Meyer, Thorsten Graber, Karen Petersen, Uwe Klingauf, and Oskar von Stryk, *Hector open source modules for autonomous mapping and navigation with rescue robots*, RoboCup 2013: Robot World Cup XVII, Springer, 2014, pp. 624–631.
18. A. Kumar and A. Vladimirovsky, *An efficient method for multiobjective optimal control and optimal control subject to integral constraints*, Journal of Computational Mathematics **28** (2010), 517–551.
19. Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun, *Ara*: Anytime a* with provable bounds on sub-optimality*, Advances in Neural Information Processing Systems, 2003.
20. E. Machuca, L. Mandow, and J.L. Pérez de la Cruz, *An evaluation of heuristic functions for bicriterion shortest path problems*, New Trends in Artificial Intelligence. Proceedings of the XIV Portuguese Conference on Artificial Intelligence (EPIA 2009)., 2009.
21. Enrique Machuca, Lorenzo Mandow, Jose L. Pérez de la Cruz, and Amparo Ruiz-Sepulveda, *An empirical comparison of some multiobjective graph search algorithms*, Proceedings of the 33rd annual German conference on Advances in artificial intelligence (Berlin, Heidelberg), KI’10, Springer-Verlag, 2010, pp. 238–245.
22. L. Mandow and J. L. Pérez De la Cruz, *A new approach to multiobjective a* search*, Proceedings of the 19th international joint conference on Artificial intelligence (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 2005, pp. 218–223.
23. Lawrence Mandow and José. Luis Pérez De La Cruz, *Multiobjective a* search with consistent heuristics*, J. ACM **57** (2008), 27:1–27:25.
24. Blane Rhoads, Suresh Kannan, Thomas A. Frewen, and Andrzej Banaszuk, *Optimal control of autonomous helicopters in obstacle-rich environments*, United Technologies Research Center, East Hartford, CT (2012).
25. J.G. Siek, L.Q. Lee, and A. Lumsdaine, *Boost graph library: User guide and reference manual, the*, Pearson Education, 2001.
26. A. J. V. Skriver and K. A. Andersen, *A label correcting approach for solving bicriterion shortest-path problems*, Computers & Operations Research **27** (2000), no. 6, 507 – 524.
27. Anthony Stentz, *Optimal and efficient path planning for partially-known environments*, Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on, IEEE, 1994, pp. 3310–3317.
28. Anthony Stentz, *The focussed d* algorithm for real-time replanning*, IJCAI, vol. 95, 1995, pp. 1652–1659.
29. Bradley S. Stewart and Chelsea C. White, III, *Multiobjective a**, J. ACM **38** (1991), 775–814.
30. Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki, *The Open Motion Planning Library*, IEEE Robotics & Automation Magazine **19** (2012), no. 4, 72–82, <http://ompl.kavrakilab.org>.
31. Ioan Alexandru Sucan, Mark Moll, and Lydia E. Kavraki, *The open motion planning library*, IEEE Robotics & Automation Magazine **19** (2012), no. 4, 72–82.

32. Chi Tung Tung and Kim Lin Chew, *A multicriteria pareto-optimal path algorithm*, European Journal of Operational Research **62** (1992), no. 2, 203 – 209.
33. Jur Van Den Berg, Rajat Shah, Arthur Huang, and Ken Goldberg, *Ana*: Anytime nonparametric a**, Proceedings of Twenty-fifth AAAI Conference on Artificial Intelligence (AAAI-11), 2011.